Institut Polytechnique des Sciences Avancées
Campus de Paris
63 bis, Boulevard de Brandebourg
94 200 Ivry-sur-Seine

Etablissement Privé d'Enseignement Supérieur
SIRET 433 695632 00011 - APE 803Z

Analyse numérique [Ma223]

TP 2 : Méthode de Cholesky pour la résolution de systèmes linéaires.

Dans ce TP, nous allons utiliser le langage de programmation Python et la librairie `numpy` afin de permettre d'appliquer l'algorithme de la décomposition de Cholesky pour la résolution de systèmes linéaires.

Léa Dupin
Aéro 2 – Classe F2

Année scolaire 2020 | 2021

Table des matières

Introduction

Contexte de ce TP

Suite directe du TP 1 dans lequel nous avons étudié la méthode de Gauss et ses variantes, ce TP 2 nous permettra d'étudier la méthode de Cholesky de la même manière ainsi que des variantes ; puis de comparer les méthodes étudiées dans le cadre de ces deux TP.

Nous allons également travailler sur les notions de temps de calcul, et de calcul de l'erreur. Dans notre cas, le temps de calcul est le temps nécessaire à l'ordinateur pour résoudre un système linéaire de taille $n * n$ avec une des méthodes étudiées. Le calcul de l'erreur sera calculé par la norme de $\|A\,X - B\|$, qui doit être au plus près de 0 pour que la méthode puisse être qualifiée de « précise ».

Nous allons donc étudier les différentes méthodes séparément, puis analyser les résultats obtenus avant de les comparer et conclure sur la plus efficace à utiliser, autant sur le critère du temps de calcul que de la précision.

Le programme informatique pour ce TP est disponible sur le site internet GitHub en suivant ce lien : https://github.com/LeaDupinAero2F2/Ma223---TP02

Petite biographie…

André-Louis Cholesky est né en 1875 en Charente-Maritime.

Il obtient son baccalauréat à Bordeaux en 1893 avec mention assez bien et intègre l'Ecole Polytechnique en octobre 1895. Deux ans plus tard, il est admis à l'Ecole d'Application de l'Artillerie et du Génie à Fontainebleau.

Il est nommé lieutenant en 1899 et effectue plusieurs missions en Tunisie et Algérie entre 1902 et 1904. De novembre 1907 à juin 1908, il effectue une mission à Crète, alors occupée par les troupes internationales. Il est nommé capitaine en 1909 et doit rejoindre le 13ème régiment d'artillerie afin d'y effectuer le temps légal de deux ans qu'il devait accomplir comme commandant de batterie. C'est à cette époque qu'il rédige un manuscrit sur sa méthode de résolution des systèmes d'équations linéaires, celle aujourd'hui appelée la « méthode de Cholesky ».

Il est décédé le 31 août 1918 des suites de blessures reçues sur le champ de bataille.

Figure 1 : André-Louis Cholesky, élève à l'Ecole Polytechnique
(reproduction faite avec l'autorisation des archives de l'Ecole Polytechnique)

Partie 1 : Décomposition de Cholesky

Soit un système $Ax = B$ à résoudre, où la matrice A est carrée, symétrique, définie positive. La méthode de Cholesky consiste alors à décomposer la matrice A en un produit $A = LL^T$ où L est une matrice triangulaire inférieure (tous les termes au-dessus de sa diagonale sont nuls) et avec des termes diagonaux strictement positifs.

Au niveau de la programmation, la démarche est la suivante :

- On créé aléatoirement une matrice pour laquelle la méthode de Cholesky fonctionnera, c'est-à-dire une matrice symétrique, définie positive. Pour cela, on utilise la propriété selon laquelle A est décomposable en deux matrices LL^T où L est une matrice triangulaire inférieure (tous les termes au-dessus de sa diagonale sont nuls) et avec des termes diagonaux strictement positifs.
- On calcule la décomposition de la matrice obtenue grâce aux formules que l'on connait.
    - Pour les termes diagonaux :

$$l_{k,k} = \sqrt{a_{k,k} - \sum_{j=1}^{k-1} l_{k,j}^2}$$

    - Pour les termes non diagonaux $i > k$:

$$l_{i,k} = \frac{a_{i,k} - \sum_{j=1}^{k-1} l_{i,j} * l_{k,j}}{l_{k,k}}$$

- On récupère notre matrice L.

Partie 2 : Résolution de systèmes à l'aide de la décomposition de Cholesky

Après décomposition, le système devient alors $LL^T x = B$. On pose $L^T x = y$ et on résout d'abord $Ly = B$ par descente, ce qui nous fournit le vecteur y. On résout ensuite $L^T x = y$ par remontée. On a donc notre vecteur solution x.

Au niveau de la programmation, la démarche est la suivante :

- On résout le système triangulaire inférieur $Ly = B$ par descente.
- On résout le système triangulaire supérieur $L^T x = y$ par remontée.
- On retourne notre vecteur solution x.

En testant l'algorithme, on obtient les temps de calcul suivants :

Taille n de la matrice carrée	Temps en secondes
50	0.03125
100	0.0625
150	0.078125
200	0.140625
250	0.1875
300	0.28125
350	0.359375
400	0.46875
450	0.578125
500	0.578125

Le temps de calcul total est de 2.1875 secondes.

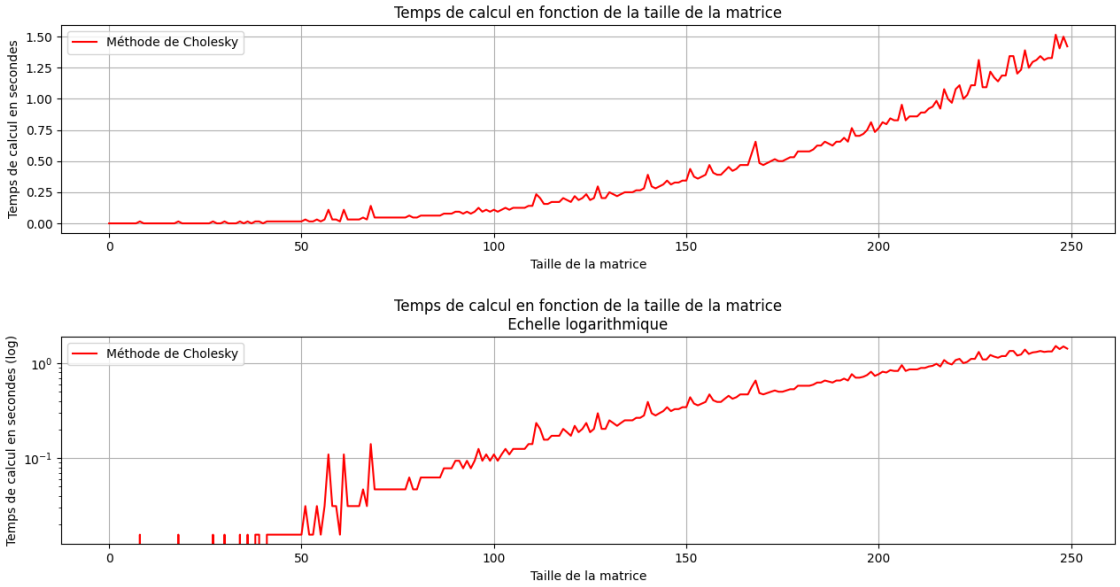


Figure 2 : Courbes obtenues pour la résolution de 250 matrices de taille comprise entre 1 et 250, par pas de 1

Le calcul de l'erreur donne ces courbes :

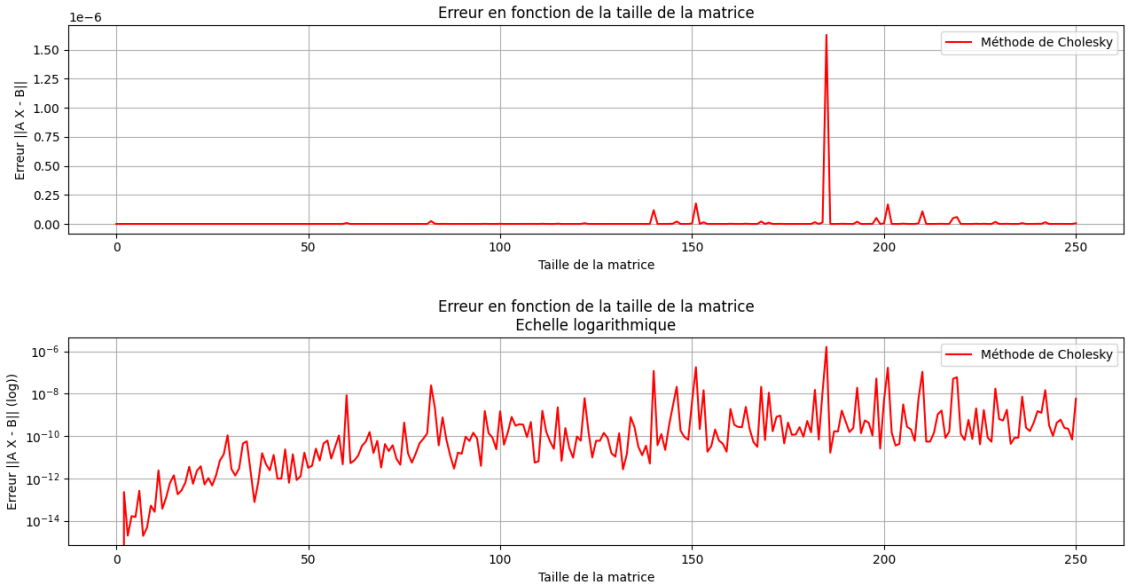


Figure 3 : Courbes obtenues pour le temps de calcul pour la résolution de matrices de taille comprise entre 1 et 250, par pas de 1.

En agrandissant la première, on a un ordre de grandeur différent (puisque l'on ne prend plus en compte les pics les plus hauts) :

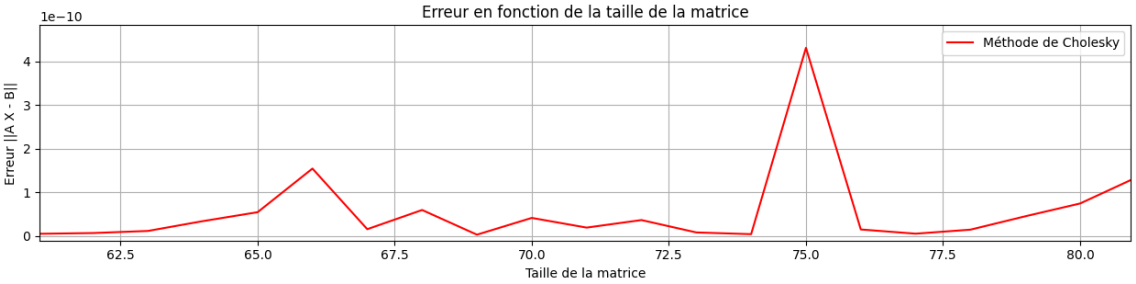


Figure 4 : Agrandi de la courbe précédente.

L'ordre de grandeur de l'erreur de calcul est donc de l'ordre de 10^{-10} ce qui est très faible : on peut donc considérer notre méthode comme précise.

Partie 3 : Expérimentation des méthodes

On souhaite maintenant comparer cette méthode avec celles étudiées lors du TP 1, c'est-à-dire la méthode de Gauss avec la décomposition LU et `linalg.solve`. On utilisera également la méthode `linalg.cholesky` qui effectue la décomposition de Cholesky, afin de comparer avec notre méthode programmée.

J'ai tout d'abord comparé les méthodes de Gauss LU, Cholesky et du solveur Python ensemble :

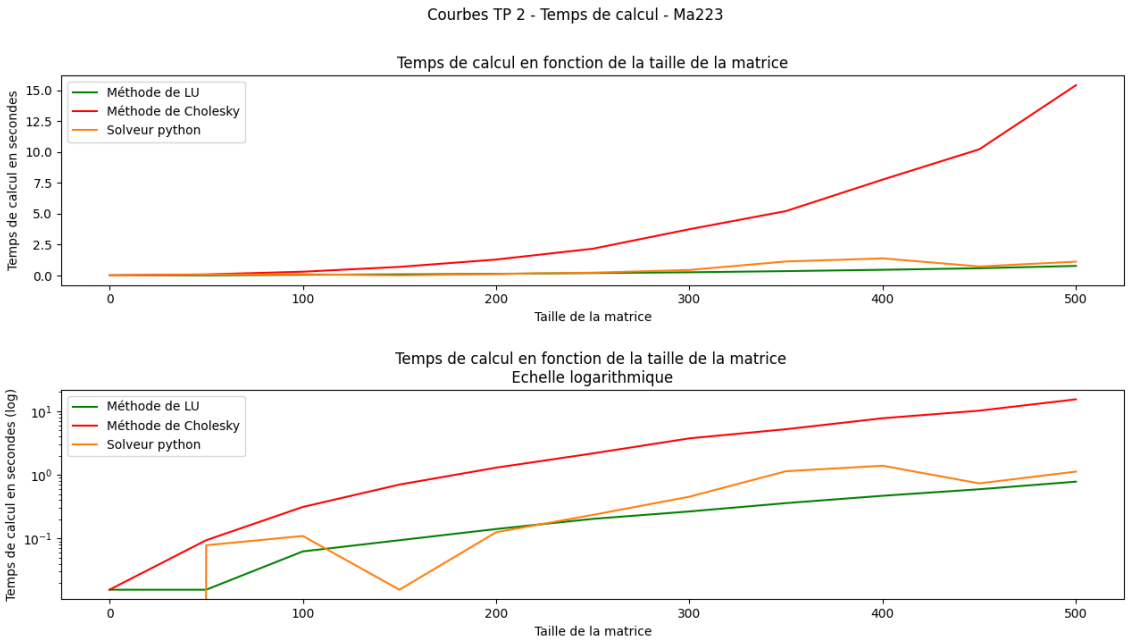
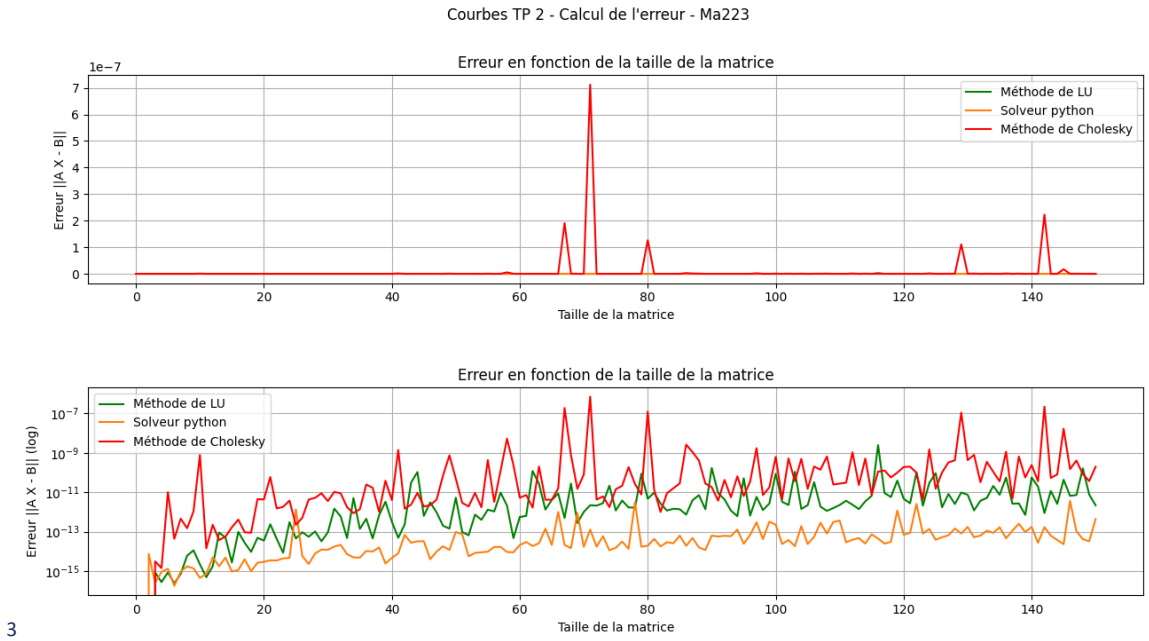


Figure 5 : Courbes de comparaison du temps de calcul pour 3 méthodes différentes.

On obtient les temps suivants :

Taille de la matrice	Temps en secondes avec la méthode de Gauss LU	Temps en secondes avec la méthode de Cholesky	Temps en secondes avec la méthode du solveur Python
50	0.015625	0.09375	0.078125
100	0.0625	0.3125	0.109375
150	0.09375	0.703125	0.015625
200	0.140625	1.296875	0.125
250	0.203125	2.171875	0.234375
300	0.265625	3.75	0.453125
350	0.359375	5.21875	1.140625
400	0.46875	7.765625	1.390625
450	0.59375	10.21875	0.734375
500	0.78125	15.40625	1.125
550	0.78125	15.40625	1.125

On peut d'ors et déjà notter que ces résultats peuvent nous sembler étrange car dans le TP 1 nous avions conclu que la méthode du Solveur Python était la plus rapide, et ici la méthode LU semble l'être…



3

Figure 6 : Courbes de calcul d'erreur obtenues pour les 3 différentes méthodes.

En agrandissant la courbe précédente, nous pouvons mieux distinguer les erreurs:

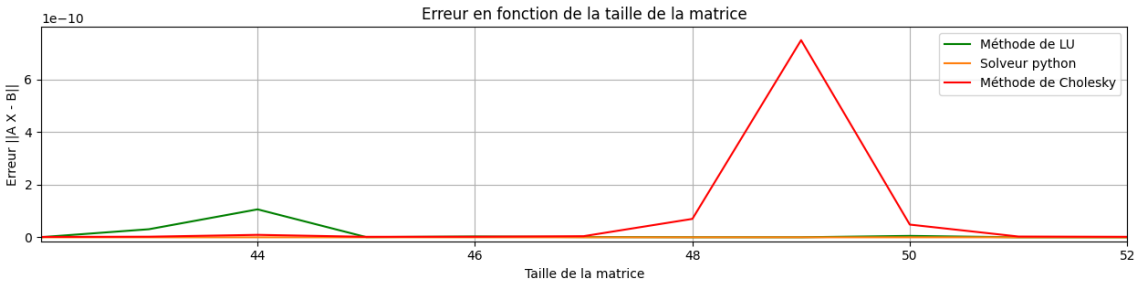


Figure 7 : Agrandi de la figure précédente.

On peut donc dire que la méthode semblant être la plus stable est la méthode du solveur Python, ce qui est cohérent en comparant avec nos résultats obtenus lors du TP 1. La méthode de Cholesky est celle qui présente le plus de pics d'erreurs.

Nous avons ensuite comparé la méthode de Cholesky telle que nous l'avons programmée avec la méthode de Cholesky intégrée au module `numpy` de python :

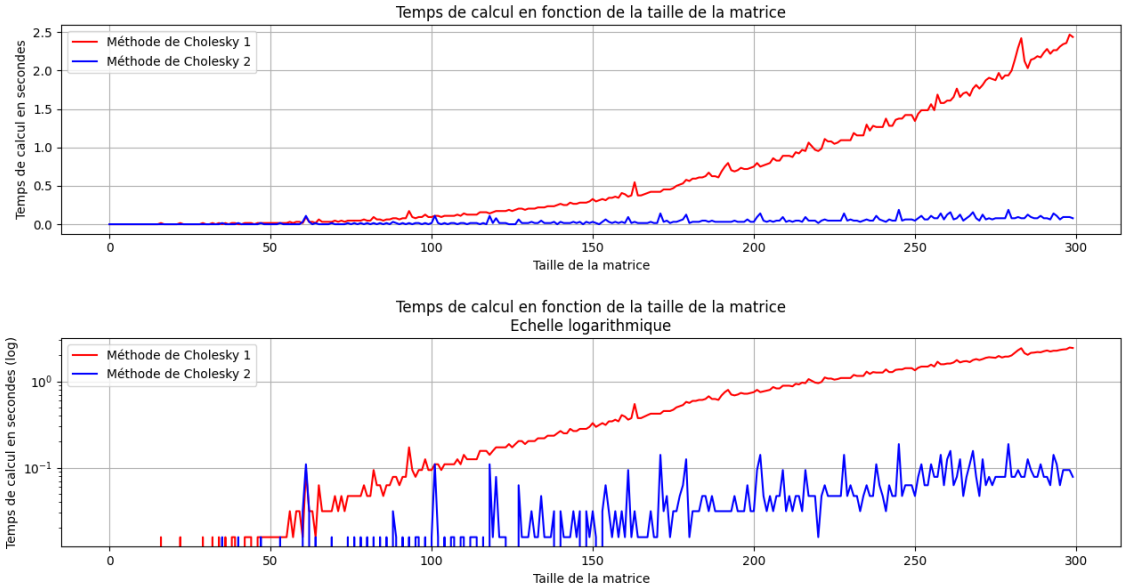


Figure 8 : courbes obtenues pour le temps de calcul des 2 méthodes de décomposition de Cholesky.

Taille de la matrice	Temps en secondes avec la première méthode de décomposition de Cholesky	Temps en secondes avec la deuxième méthode de décomposition de Cholesky
50	0.03125	0.0
100	0.09375	0.015625
150	0.34375	0.03125
200	0.75	0.03125
250	1.46875	0.0625
Temps total en secondes	**94.421875**	**6.40625**

On remarque que la méthode 2, celle du `linalg.cholesky` est beaucoup plus rapide que celle que nous avons programmée. On peut imaginer que cela est dû à l'optimisation du code.

L'erreur de calcul est la même puisque le calcul en lui-même est le même. Il n'y a que la méthode de décomposition qui diffère donc l'erreur reste inchangée.

On compare donc à présent les deux méthodes les plus rapides que nous ayons : le solveur Python et celle du `linalg.cholesky` :

Taille de la matrice	Temps en secondes avec la méthode `linalg.solve`	Temps en secondes avec la méthode `linalg.cholesky`
50	0.015625	0.015625
100	0.0	0.0
150	0.0	0.078125
200	0.046875	0.015625
250	0.015625	0.0625
300	0.046875	0.09375
350	0.15625	0.140625
400	0.109375	0.15625
450	0.234375	0.234375
500	0.15625	0.296875
Temps total en secondes	**37.703125**	**50.421875**

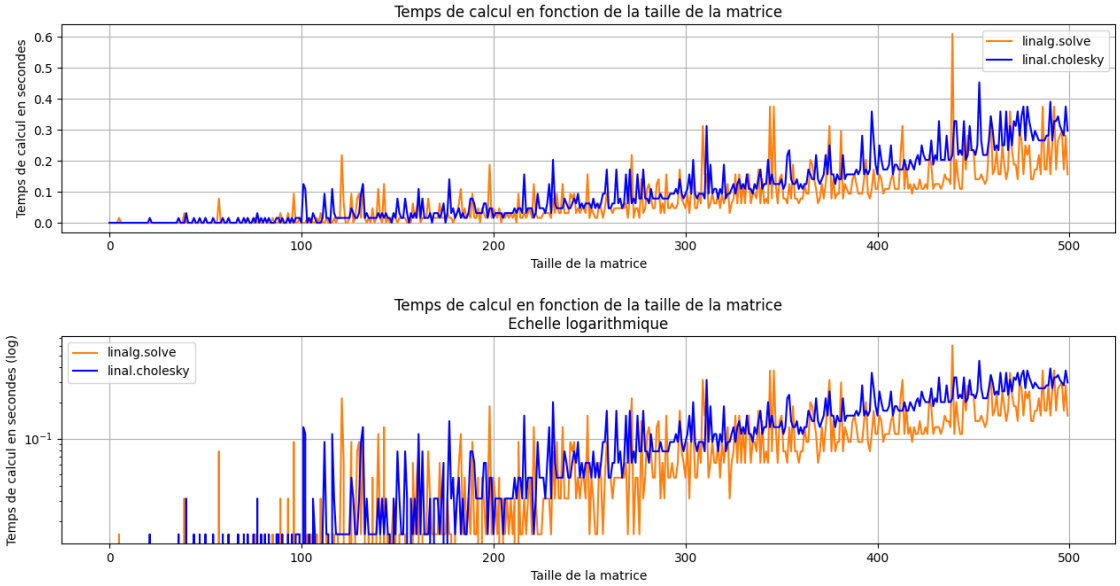


Figure 9 : Courbes obtenues en utilisant les méthodes intégrées au module `numpy` de Python.

On peut dire que ces deux méthodes sont très proches en termes de temps de calcul. Cependant, ce qui pourra faire la différence sera le calcul de l'erreur.

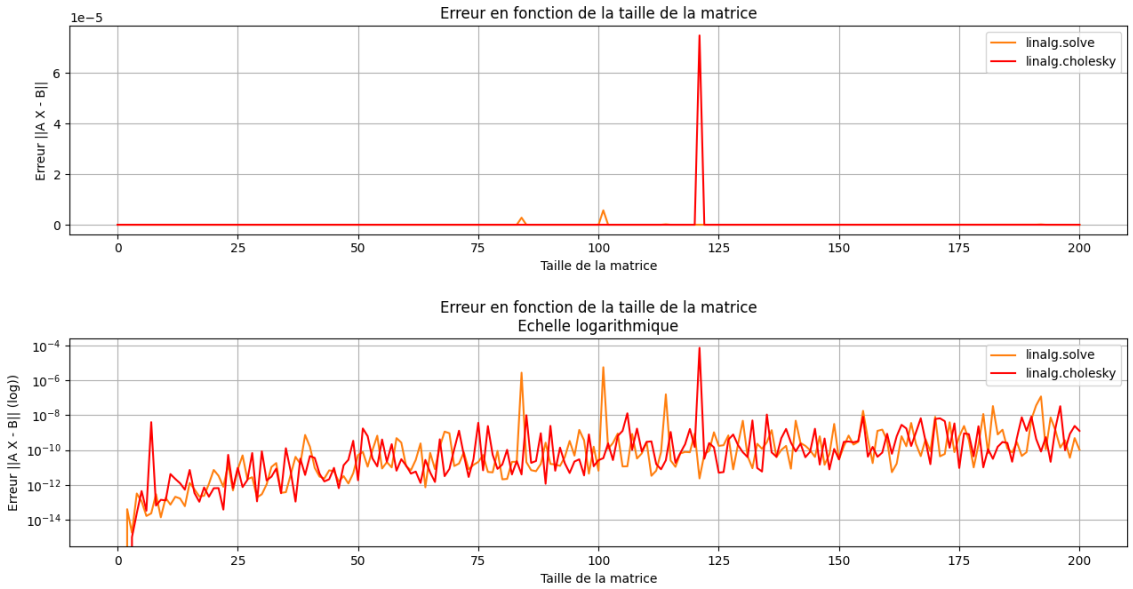


Figure 10 : Courbes de calcul de l'erreur avec les deux méthodes intégrées dans Python.

En agrandissant le graphique précédent on obtient :

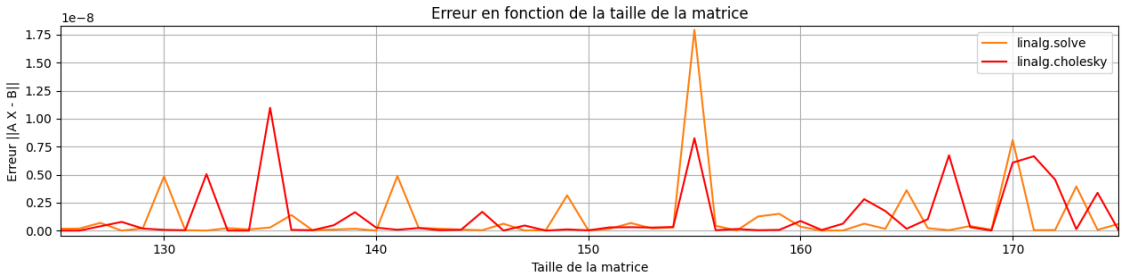


Figure 11 : Agrandi de la figure précédente.

Finalement, les erreurs sont assez semblables également, malgré plus de gros pics avec la décomposition de Cholesky. Peut-être pouvons-nous supposer que la méthode de solveur de `linalg.solve` utilise cette décomposition ?

On terminera nos comparaisons par les trois méthodes les plus efficaces jusqu'ici : Gauss LU, `linalg.solve` et `linalg.cholesky`.

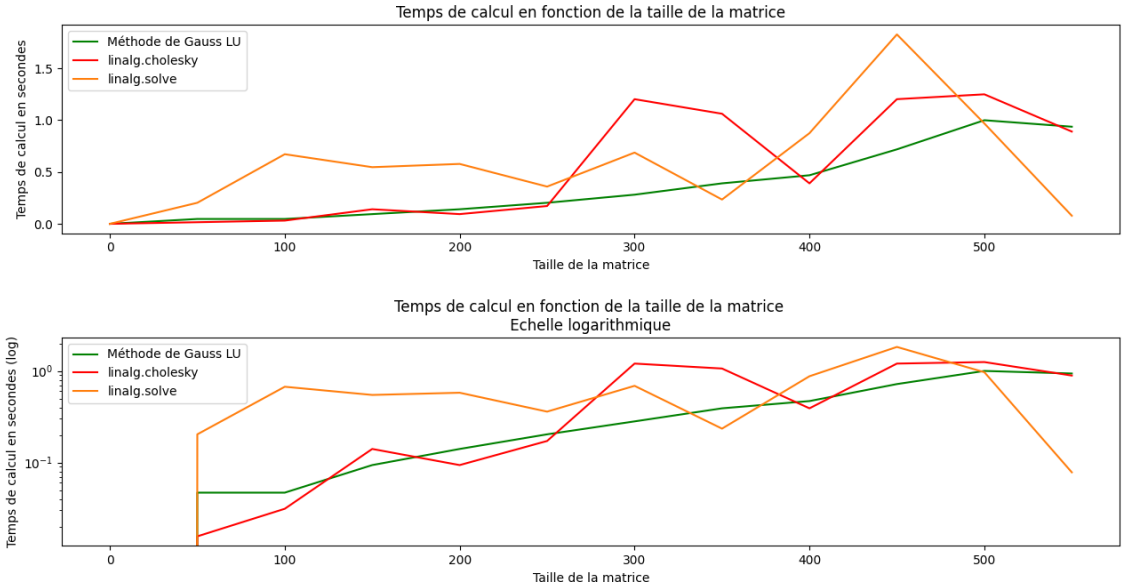


Figure 12 : Courbes obtenues du temps de calcul en comparant les 3 méthodes les plus efficaces.

Le temps de calcul est similaire pour les 3 méthodes.

Figure 13 : Courbes du calcul de l'erreur.

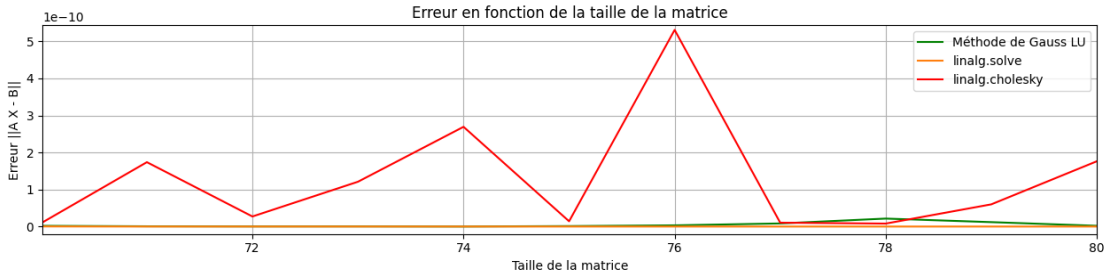


Figure 14 : Agrandi de la figure précédente.

Les erreurs sont semblables aussi, malgré une stabilité moindre pour la méthode utilisant le `linalg.cholesky`, notamment à cause de la façon dont nous avons programmé cette méthode.

Conclusion

Si nous devions donc choisir entre les différentes méthodes que nous avons étudié au cours de cet enseignement de Ma223, nous devrions alors préférer la méthode du `linalg.solve`, plus de Gauss LU et enfin de `linalg.cholesky`.

Ces trois méthodes sont les plus optimisées en termes de temps de calcul et de faible marge d'erreur.

Annexe : Méthode statistique pour lisser les courbes d'erreurs de calcul

Nous avons étudié la possibilité de calculer un nombre n de fois la résolution d'un système de taille N, puis de moyenner les erreurs calculées, afin d'éviter les grands pics d'erreurs que nous pouvons voir sur chaque graphique. Cependant, pour ma part, les résultats n'ont pas été concluants notamment à cause d'une erreur de programmation entrainant l'arrêt du processus. Pour les fois où cette méthode statique a fonctionné, nous avons obtenu des résultats corrects, avec une diminution visible des pics brutaux d'erreur :

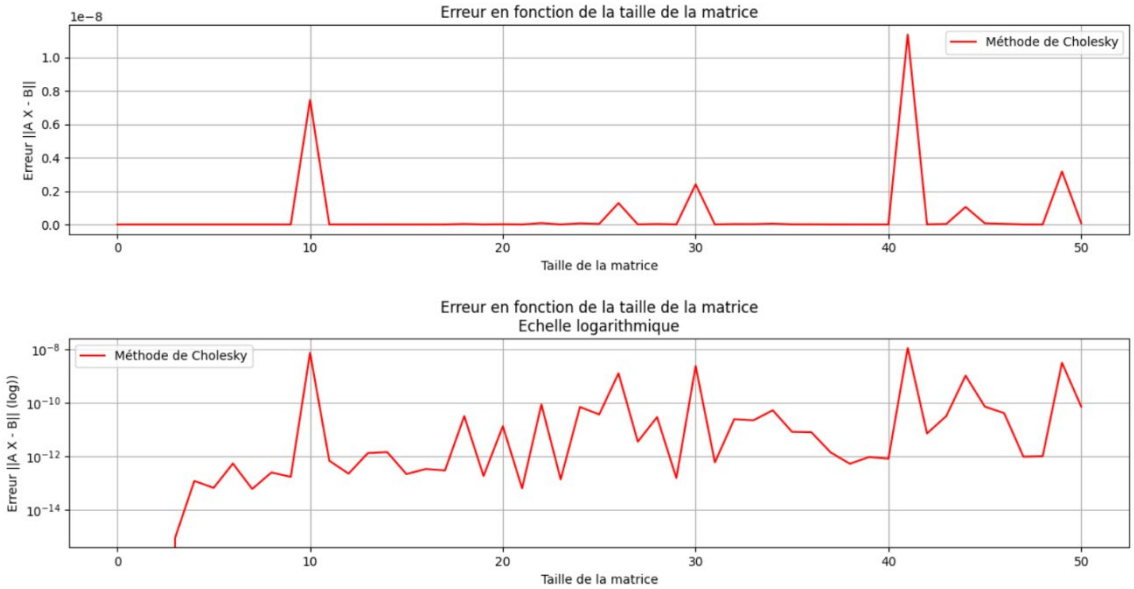


Figure 15 : Méthode classique de calcul de l'erreur.

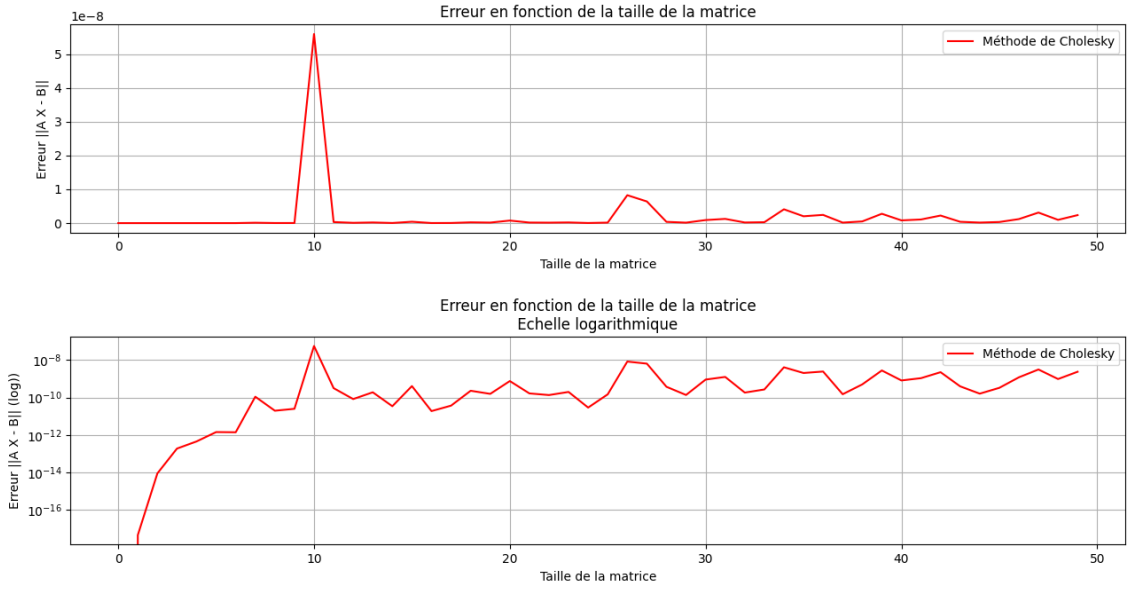


Figure 16 : Méthode statistique de calcul de l'erreur.